Model-Driven Architecture Based Security Analysis

Saoussen Mili, Nga Thi Viet Nguyen, Rachid Chelouah

ORIGINAL ARTICLE

Journal Section

Model-Driven Architecture Based Security Analysis

Saoussen Mili[1] | **Nga Nguyen**[1] | **Rachid Chelouah**[1]

[1]ETIS Laboratory, Cergy, France

Correspondence
Nga Nguyen
Email: nga.nguyen@eisti.eu

Funding information

This paper proposes a Model-Driven Architecture approach for the development of an embedded system validation platform namely Model-Based Security Analysis for Embedded Systems (MBSAES). The security properties are formally modeled and verified at an early stage of the design process of the system, which helps to reduce late errors and development time. A separation of the attack scenarios and the system design from the implementation details has been respected. To transform semi-formal models from SysML to NuSVM model checking platform, two Model-to-Text, horizontal and exogenous transformations have been implemented. The first one employs a programming approach with Java to create a Computational Tree Logic specification from an Extended Attack Tree, whereas the second one uses a template approach with Acceleo to generate NuSMV code from SysML structural and behavioral models. To illustrate our approach, a case study, involving attacks aiming to unlock car door systems, via signal jamming and code replaying, is considered. The results of this research will contribute to the automatic validation of system designs against security vulnerabilities via a database of extended attack trees building from existing atomic attacks.

KEYWORDS
MDA, SysML, model transformation, code generation, embedded systems, security, model checking

1 | INTRODUCTION

In 2001, the Object Management Group (OMG) launched **Model-Driven Architecture** (MDA), an architectural framework for software development to build portability, interoperability and re-usability [1]. The approach consists of separating the business logic of the company from any technical platform, the need for separation being justified by the fact that the business logic remains fairly stable during the product life cycle and undergoes little modification, unlike the technical platforms. MDA provides a set of modeling and transformation techniques. It offers three default models: **Computational Independent Model** (CIM), **Platform Independent Model** (PIM) and **Platform Specific Model** (PSM). CIM describes the function of a system, i.e. the system's environment and the requirements, without showing constructional details. PIM details the construction of the system, i.e. composition, structure, etc. without implementation details. Lastly, PSM defines the technical details related to the implementation of PIM specifications within a particular type of platform. Besides these models, MDA also specifies a **Platform Model** which characterizes the technical concepts such as parts of the platform and the services provided, containing information for transforming models to a specific platform. Summing up, the MDA approach allows generating code associated with software, which is PSM, from the CIM and PIM, using a transformation platform model. This process of transformation can save time and improve robustness to software, in particular to those endowed with great complexity [2].

Depending on the source model type to transform and the destination model type to generate, we distinguish three types of model transformation processing : **Model-to-Model** (M2M), **Model-to-Text** (M2T) and **Text-to-Model** (T2M). In the literature, there are several approaches for classifying model transformations such as in [3] and [4]. A transformation can be endogenous or exogenous. A transformation is said to be **endogenous** when the meta-models for source and target models are identical, while it is said to be **exogenous** when they are different. Another classification in terms of level of abstraction is offered, the **horizontal** versus **vertical** transformation. The vertical transformations of models are used to refine or abstract models and, in this case, the models are located in different levels of abstraction. Horizontal changes do not affect the abstraction models and they are mainly used to restructure, since these models belong to the same level of abstraction. The languages dedicated to the transformation of models can be declarative, imperative or hybrid. A **declarative** language describes the meta-models to process and constraints on these meta-models, by focusing on the *what* aspect. An **imperative** language describes *how* the result is achieved by imposing a series of elementary and detailed actions that the transformation performs. A **hybrid** language is both declarative and imperative.

According to [5], there are several model transformation approaches, namely programming approach, template approach and modeling approach. In the **programming approach**, the idea is to program a model transformation in the same way as any software application, using typically object-oriented languages, and the data structures are the models involved in the transformation. The **template approach** defines template models and replaces their parameters by the values of the source models. Parameterized target models are template models containing parameters, which are substituted by the values of the source models, at a subsequent stage, during processing. This approach uses specific languages for the definition of template models. By analogy to the MDA, the **modeling approach** is shaped as transformation rules based on Model-Driven Engineering, a software development methodology focusing on models. In this approach, a model of transformation is structured by its meta-model, and the models are independent of execution platforms. The objective of this approach is to perpetuate the transformations models and make them productive.

Our work is contextualized in the field of automatic code generation for security analysis on communicating embedded systems. The choice of this type of system is justified on the one hand by their safety-critical and security issues and on the other hand, by the number of attacks that they have suffered in recent years, for example some

famous remote controls takeover of connected cars. In our previous work [6], we proposed an approach allowing the verification and validation of complex embedded systems against the propagation of attacks. This approach is composed of three steps: 1) attack modeling with the Extended Attack Tree Profile using temporal logic operators; 2) structural and behavioral system modeling with an appropriate Connectivity Profile to enrich system models with error propagation through components; and 3) formal validation of systems against vulnerabilities via a model checker, thanks to automatic code generation from system models. The overall method is named Model-Based Security Analysis for Embedded Systems (MBSAES). To represent the semi-formal models in steps 1 and 2, the Systems Modeling Language (SysML) [7] is used to introduce Extended Attack Trees (via Activity Diagrams) and to describe the system static and dynamic points of view, through Internal Block Diagrams (IBD) and State Machine Diagrams (SMD) respectively, whereas, the NuSMV language [8] is chosen for the model verification in step 3.

The objective of this paper is to describe our MBSAES approach by using MDA notions. The elements of the framework have already been introduced in [6] but MBSAES is represented in this work with a completely new perspective, based on MDA viewpoints. With the benefit of hindsight, these elements are clearly defined with respect to CIM, PIM and PSM. Model transformations, i.e. attack encoding and system transformation within the Platform Model, are detailed using OMG meta-model layers. With respect to the state of the art, MBSAES brings together SysML models developed with specific profiles (temporal representation of attacks and connectivity of systems) with model checking, via automatic model transformations. The main addressed research question is "how this modeling methodology, based on model transformations and formal verification, can support security analysis to detect potential security problems in an effective way?". Separating attack scenarios and system description from implementation details allows applying automatically formal methods, a discipline which is not easy to comprehend for systems engineers, to system security analysis. Model transformations enhance the updates of model checking code according to changes in system design with minimal time and effort. The scope of MBSAES is to help systems engineers and security engineers to validate or discard a design solution for connected and embedded systems, from a security perspective, in the system design processes. The paper is structured as follows. Section 2 describes some work related to model-driven security analysis. Section 3 details our MDA application to security analysis. The approach is illustrated via a case study on attacks that try to unlock car doors in Section 4. Discussions and conclusion are given in Section 5.

2 | RELATED WORK

All damage caused by security breaches argues for an effective security policy. A policy that is able to adapt to the complex nature of safety-critical embedded systems, on the one hand, and which adjusts and responds effectively to rapid system changes, on the other. The literature reviewed in this section is selected with respect to Model-Based Engineering, the engineering at the model level, an approach that allows, in principle, to tackle the complexity of Cyber-Physical Systems (CPS) via security by design, abstraction, and automation, as analyzed in [9].

Basin et al. in [10] count among the first groups to introduce the Model-Driven Security (MDS) concept to solve this problem. They proposed a UML profile called *SecureUML* to represent the formalization of Role-Based Access Control (RBAC) requirements into design models. They established that integrating security properties within models reduces the gap between system design and implementation, thus facilitating system maintenance and evolution and enhancing portability, since migration to new technologies became simpler. An example of *Order placing* is used to describe the importance of security by adding features like access control policy and action hierarchies in the model to grant access.

In [11], the authors Jürjens and Shabalin presented *UMLsec*, an extension of UML, designed for integrating the specification of security access control into application models. UMLsec defines a vocabulary for expressing different aspects of access control, e.g., *roles*, *role permissions* and *user-role assignments*, to assist the development of security-critical systems. ArcStyler is used as a modeling and code generation tool, and an automated verification of UMLsec diagrams is carried out via the SPIN model-checker [12]. This technique only targets software development, while the complex architecture of CPS requires an adequate language able to model the different parts of such a system and, in particular, the interactions between them.

The authors in [13], Fernández-Medina and Piattini, used MDS for the design of secure databases. Models have been proposed to include security information in the database model, as well as a constraint language to define security constraints. The proposed approach was applied to a real case *Data Processing Center of Provincial Government* by using UML profiles.

Reznik et al. [14] presented a MDS approach on middleware platforms to develop secure applications which integrate an implementation of the Corba Component Model with the OpenPMF security framework. An example of platform independent definition of Air Traffic Management system, in conjunction with security policies, is used to explain how the automatic transformations of complex systems with security definitions into platform specific artifacts reduces the development time.

Another MDS approach called *ModelSec* proposed by Sánchez et al. in [15] offered a generative architecture to manage security requirements, going from the definition of the requirements to the implementation. This architecture automatically generates security requirements using two model transformation chains. It also ensures the separation of security requirements and design choices. ModelSec conceived a high-level automation for code generation that aims to deal with security specifications of the system in order to provide easier, quicker, more efficient and error-free process. A M2M transformation is used for generating software artifacts related to system security. An example from the paper shows that healthcare security software artifacts are automatically generated from the target platform model of the security design models representing low-level data of the access control policy, the security database and the target platform by using the eXtensible Access Control Markup Language (XACML). A web application example for the management of medical patients which includes the design of a secure database shows how a role-based control access policy and database can be automatically generated. The public, confidential and sensitive elements have been added to the model.

In [16], Delange et al. proposed an approach to design and validate safety-critical systems that uses Architecture Analysis and Design Language (AADL) [17] to model partitioned architectures with their requirements and properties. The models are validated through the PolyORB Kernel (POK) Checker. Use-cases have also been proposed to demonstrate the correctness of the implementation and are available in the releases of the POK project.

Ouchani et al. [18] presented a formal framework to perform security risk assessment and security requirements verification, by using SysML Activity Diagrams to model systems. The behaviour of attack scenarios specific to a Real Time Streaming Protocol (RTSP) application is automatically translated into PRISM code for a formal verification. As a result, probabilistic model-checking helps to reduce the cost of software development as errors are detected at an early stage.

In the work of Mehrdad et al. [19], the Modeling and Analysis of Real-Time Embedded Systems (MARTE) language [20] is extended with security aspects that represent embedded system security requirements. An example showing how to model encryption requirements of the *AuthorizePayment*() operation has been proposed. A stereotype called *Encryption* is also proposed. Another example yet of vehicle authentication is presented. It consists in storing unique identification information securely. A stereotype for secure memory modules is defined. The proposed security profile considers different classifications and taxonomies for security such as logical security, hardware security and

security incidents.

In [21], Apvrille and Roudier proposed SysML-Sec, an environment to design safe and secure embedded systems in which attacks are described by using an extension of SysML Parametric Diagram. The threats are modeled as attributes embedded into blocks representing the target of the attacks. The attacks can be linked together via constraints in timed automata to model the timing behavior. The operators are either logical operators like AND, OR, and XOR, or temporal causality operators like SEQUENCE, BEFORE and AFTER. Formal verification about the reachability or the liveness of an attack and the "leads to" relationship between two attacks can be carried out by using the Uppaal model checker [22]. They also used the free and open-source software TTool [23], which supports simulations and formal proofs on SysML models, for both safety and security properties, right from the architecture mapping and software hardware partitioning phases. The formal security proofs on authenticity and confidentiality of SysML-Sec designs rely on ProVerif [24], a tool for cryptographic protocol analysis based on Horn clauses. However, since the time concept is not supported by ProVerif, the simplified translation to ProVerif specifications could lead to some false positive results [25].

Mota et al. [26] proposed a new model checker with the COMPASS Modeling Language (CML) whose purpose is to provide support for modeling system-of-systems with infinite aspects. This model checker is embedded into the Symphony IDE, which is integrated with a SysML tool (Artisan Studio), so CML model files can be generated directly from SysML models. However, no model transformation was detailed in this work, and the formal verification concerns safety properties rather than security requirements.

In comparison to related work, such as those reported above, the solution we devised allows refining system models with specific properties using the "extension by profiling" mechanism. It covers both structural and behavioral aspects, and can be applied at both software and system levels. Furthermore, to leverage the shortcomings of other attack representation approaches, we proposed an extended attack tree format with temporal logic operators that represent the dynamic interactions between the atomic attacks. It is also worth noting that the model transformations and automated formal verification in these model-based security studies were used to a rather limited extent. The main improvement of MBSAES is to bring together the profiles developed in SysML, the model transformations and automatic model checking. In addition, a library of extended attack trees will be generated from existing atomic attacks that allows automatic validation of system design with respect to security vulnerabilities. Real-life attacks have been modeled, and a transformation platform has been implemented to prove the feasibility of the approach, as described in the next section.

3 | FROM SYSTEMATIC ARCHITECTURE TO SECURITY ANALYSIS

This work focuses on a model-based approach for an early system validation, right from the design process, with respect to security requirements. We proposed a formal validation approach based on MDA by, during the design process, separating the domain aspect and the technical aspect. Therefore, our MDA process is composed of an iterative design series of CIM, PIM and PSM. The CIM models contain informal descriptions of different attack scenarios and the studied system which will be specified afterwards with PIM models by using appropriate SysML profiles. Since we carry out the transformation from SysML models to NuSMV models, the PIM models are represented by the SysML models, and the PSM models are constituted by the generated NuSMV code and Computational Tree Logic (CTL) [27] specifications. M2T transformations [28] are executed for the automatic code generation. Figure 1 gives the corresponding MDA models with respect to our MBSAES methodology. Each of these models is described in the following

sections.

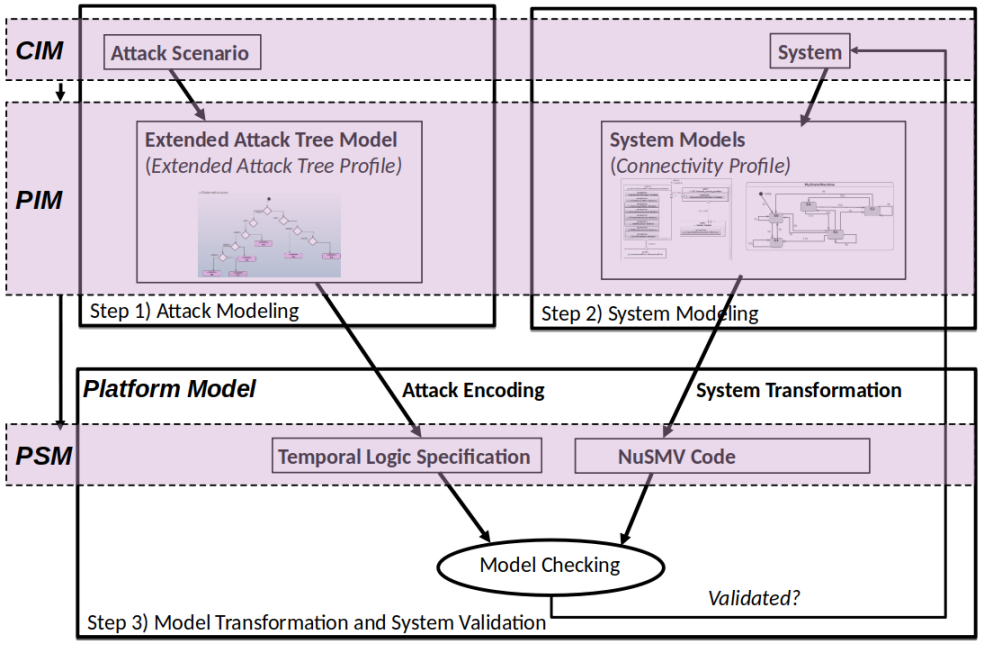

FIGURE 1 MDA-Based Security Analysis

3.1 | Computation Independent Model (CIM)

The CIM is constituted by the attack scenarios and the system description.

3.1.1 | Attack Scenario

The informal attack specification represents the security requirements of the system. This attack scenario, called global attack, is divided into a series of atomic attacks. The latter can be considered as an attack attempt that corresponds to an indivisible simple action as defined in [29]. These atomic attacks have the specificity to be independent of each other. The level of attack decomposition is a choice made by the designers and justified by their desired level of abstraction.

3.1.2 | System

The system description can be specified by the business analyst in cooperation with the business user. It represents the system of interest, its boundary as well as its environment. The system will be transformed into a PIM by a system engineer or an enterprise architect in the next step.

3.2 | Platform Independent Model (PIM)

Our PIMs contain the attack and the system models that are described respectively in the next subsections.

3.2.1 | Extended Attack Tree Model

As described in [6], to model the attacks, an *Extended Attack Tree Profile* is integrated to introduce the notion of time using temporal operators from CTL. In temporal logic, operators such as Globally (G), Future (F), Next (X) and Until (U) enable the description of the way conditions change over time in distributed systems. With CTL, path quantifiers such as A (for all paths) and E (there exists a path) can be used to express some safety properties, such as a bad thing never happens (AG not bad-thing) or a bad thing could happen (EF bad-thing). We propose to extend the $UML4SysML :: ControlNode$ meta-class by adding stereotypes which represent CTL operators, and also to extend the $UML4SysML :: Action$ meta-class by introducing a set of atomic attacks. Figure 2 shows the structure of the Extended Attack Tree Profile.

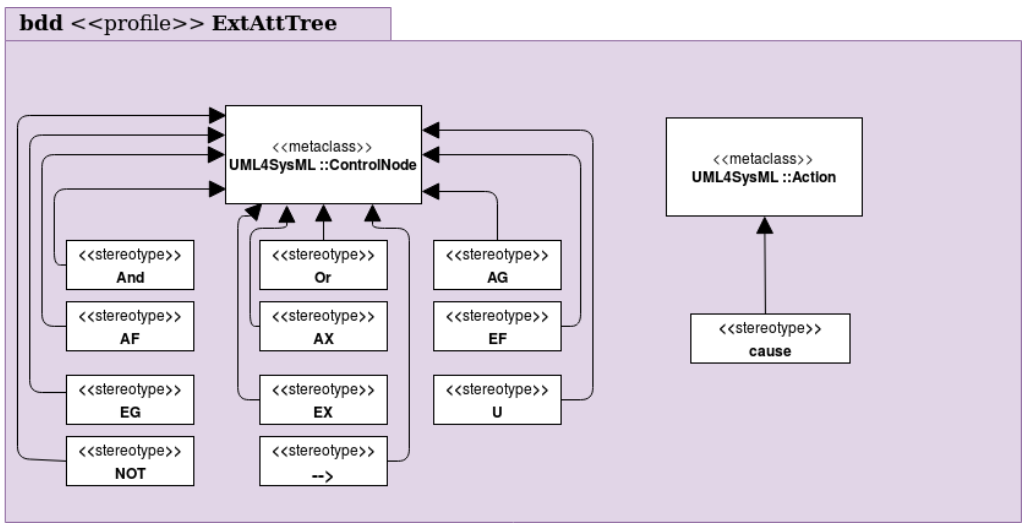

FIGURE 2 Extended Attack Tree Profile [6]

We can then define an extended attack tree $Attack(L, N)$ where L is the set of leaves, and N the set of nodes. The nodes correspond to two types of operator: unary operators like NOT, AG, AF, AX, EG, EF, EX, and binary operators, such as AND, OR, U, $\rightarrow$. The leaves represent the operands that specify the attacks. To represent graphically our extended attack trees, we use SysML Activity Diagrams. An example is given in Section 4.

3.2.2 | System Models

In this step, the system is modeled with the semi-formal language SysML. For this purpose, two types of diagrams are used: IBD, which covers the structural aspect, and SMD, which represents the behavioral aspect. At this design level, architecture modeling consists in representing the system, its components, and the interactions between them. IBD describes the internal architecture of the system in terms of parts, ports and connectors. The system is represented

by a block, which can be made up of parts with varying depth levels. Parts are linked via a connector. We can also apply a multiplicity at the end of the connector to define the number of instances associated with the links. However, the latter does not reveal anything about the nature of the interactions. In order to identify the propagation of attacks in communicating systems, a SysML profile, named *Connectivity Profile* and aimed at modeling the connectivity properties between the components of the system, is integrated. This profile is represented in Figure 3, listing the default values of certain communication technologies. The profile extends the $UML4SysML :: Connector$ meta-class to other properties and methods such as scope, bit rate, frequency, energy, number of supported nodes, etc. SysML has two types of ports: *flow ports* that allow the flow of data and *standard ports* that are suitable for invoking services. The $SysML :: Ports\&Flows$ meta-class has also been extended to represent properties such as port state, port authentication, update state and access control. The profile has a dynamic character, allowing stereotypes to be enriched according to the nature of the modeled environment (connected cars, home automation, etc.) with heterogeneous connections (CAN, WIFI, Bluetooth, NFC, ZigBee, ...).

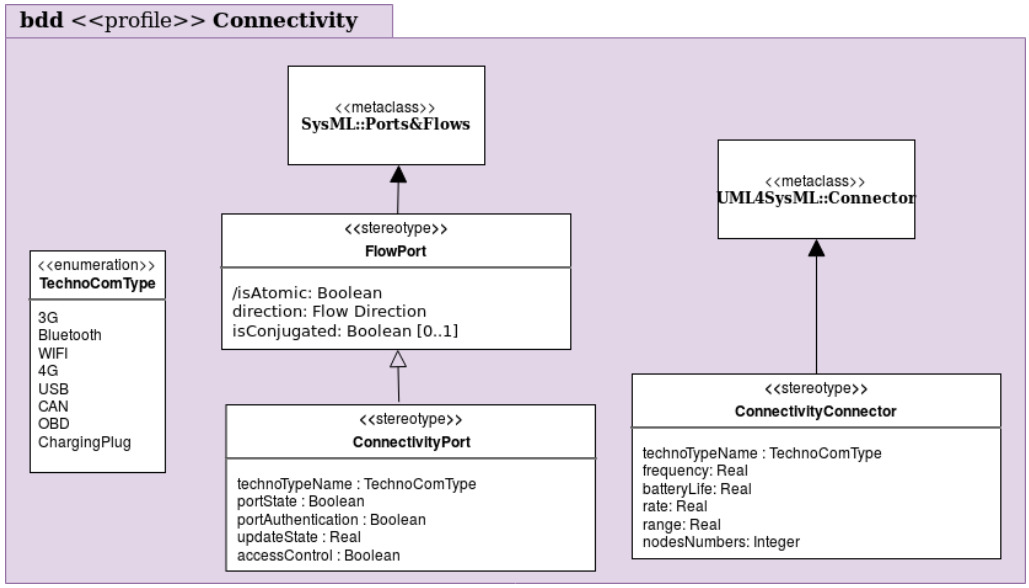

FIGURE 3 Connectivity Profile

During its life cycle, an instance of a block or a part can have a succession of states. The maintenance or change of state is generally conditioned by the satisfaction of a condition or the triggering of an event. Conditions and events are often composed of parameters, which materialize the flow of information or the data received by a component. An illustrative example with an SMD is given in Section 4 to model the dynamic behavior of the system.

3.3 | Platform Model

Our M2T transformations are classified as exogenous and horizontal. First, the PIM model of the attack, which is an extended attack tree, is transformed into a CTL formula, which represents the simulation model for the attack. This transformation uses a programming approach, with Java as the implementation language. Secondly, another template-based transformation generates a simulation model for the system, in the form of NuSMV code, from a PIM model,

which is represented via IBDs and SMDs.

3.3.1 | Attack Encoding

To generate the temporal logic expression of an attack from the Extended Attack Tree model, we used a recursive algorithm, which is an in-order depth-first search. The tree traversal starts from the root of the attack tree, and then concatenates the sub-expression of the right child with the node expression and the sub-expression of the left child. The algorithm is implemented in Java. The transformation of the conceptual model into a simulation model takes place on the M3 level, as shown in Figure 4. The meta-meta-model M3 corresponds to the highest level of abstraction in the OMG meta-model hierarchy, with items at each lower level being dependent upon the items at the upper level. Thus, an object diagram at the M0 level is defined by a user-defined model at the M1 level, which in turn is defined by a meta-model at the M2 level.

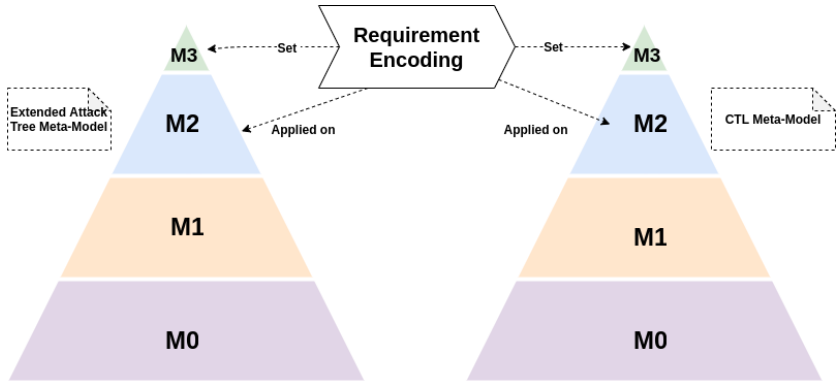

FIGURE 4 Security Requirement Encoding

The function, which makes it possible to translate a formal model into another formal model, is called encoding. Unlike a transformation, the encoding ensures the integrity of the information, i.e. no information is lost during the transformation. The encoding follows the meta-model mapping given in Table 1.

TABLE 1 Mapping between Extended Attack Tree and CTL Specification

Extended Attack Tree	CTL Specification
Atomic attack / Leave	Atomic proposition
Operator / Node	CTL operator
Binary tree	Priority and scope of operators

3.3.2 | System Transformation

In [3], the authors described the transformation rules allowing the mapping from a source model to a destination model, by formalizing certain correspondences between the two models. The transformation process therefore makes

it possible to generate the Right-Hand Side variables from the Left-Hand Side ones. These elements are described in Table 2 for our SysML and NuSMV mapping.

TABLE 2 SysML and NuSMV Mapping

SysML Meta-model	NuSMV Meta-model	SysML Meta-model	NuSMV Meta-model
Block	Module	State	State variable
Part	Module	Initial state	init
System	Main module	Condition	Case
Value properties	Variables	Transition	Transition
Input/output port	Input/output variables		

Our transformation is based on the MOF Model-to-Text Transformation Language (MOFM2T) standard [30]. We used Acceleo [31] which is an Eclipse implementation of the Model Transformation Language (MTL). Acceleo adopts an imperative executable logic. As shown in Figure 5, the transformation rules are established at the M2 level between the meta-models and executed at M1 level on the models.

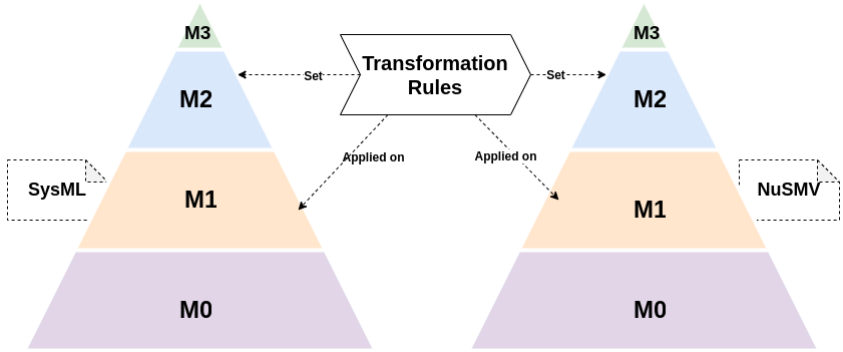

FIGURE 5 System Transformation

3.4 | Platform Specific Model (PSM)

These PSMs correspond to the temporal logic specification and the NuSMV code. A model checker will be used to verify whether the finite-state model of the system (represented by the NuSMV code) meets the required specification (represented by the CTL formula).

3.4.1 | Temporal Logic Specification

Once defined, a CTL specification will be validated or not by a marking algorithm that carries out an exhaustive state-space exploration, using techniques like breadth-first search in a graph traversal. The algorithm verifies for every state of the Kripke structure [32] that corresponds to the NuSMV code if the sub-formula of the specification is satisfied. The use of formal methods is very useful since it offers an automated verification of the specified properties and also

provides counter-examples (a path or a sequence of steps leading to the undesired state) when the properties do not hold.

3.4.2 | NuSMV Code

A NuSMV code is made up of $Modules$. Each module is associated with a finite state machine which can range from a synchronous Mealy machine to an asynchronous network of non-deterministic processes [33]. A module can contain *sub-modules*, which allows handling complex systems. We can define modular hierarchies and reusable components with the NuSMV language. A module contains two parts. The first part, specified by the keyword VAR is dedicated to the declaration of variables (what we will call atomic propositions) and the declaration of states. As for the second part, it is determined by the keyword $ASSIGN$. It allows, on the one hand, the initialization of the variables thanks to the *init* instruction and, on the other hand, the description of their respective evolution as well as the representation of the state transitions.

4 | CASE STUDY

In this section, we apply our MDA-based security analysis to a case study which is car unlocking attacks described in [34]. This example is selected because the attacks happened in an order that can be suitably specified by temporal logic. This section is arranged according to the three steps of MBSAES described in Figure 1. Thus, subsection 4.1 presents CIM and PIM models for the attacks, whereas subsection 4.2 details CIM and PIM models for the system of interest. The latter is composed of the car central locking sub-system, the doors and the remote key, as well as the connections between them. Lastly, subsection 4.3 unveils the Platform Model as well as PSM models.

4.1 | Attack Modeling

Attacks targeting central locking systems have occurred in different scenarios, which are described as follows.

- The attacker can interfere with the owner-emitted radio frequencies when the user locks the car. This will prevent the locking and therefore leave the vehicle open.
- The attacker can stand as close as possible to the owner of the key, by having identified the owner beforehand, and then activate a powerful radio repeater: the key signal can then be repeated and reaches the vehicle even if the owner gets some distance away.
- The attacker can intercept the radio signal sent by the remote key to the car and save the associated binary diagram. There are then two distinct sub-cases :
  - If the encryption is based on fixed code, the attacker can reuse this signal when he wants ans as many time as he wants;
  - If the encryption is a rolling code (a rolling code transmitter consists of a fixed part and a randomly generated part and the latter is always different from its previous ones), the attacker must initiate an interference after intercepting signals. There exist tools available on the market under the name of "rolling codes cracker".

The attack scenarios on fixed code are easier to carry out, since it takes only about 8 seconds to brute force an 8-to-12 bit code, as demonstrated in the OpenSesame attack using De Bruijn sequence [34]. Actual vehicles use

key fobs, which are equipped with a small security hardware device with built-in authentication used to control and secure access. With this rolling code policy, the access code is generated randomly, and changes periodically, so replay attacks are prevented. In order to gain access to the car, jamming technique could be used when the user presses the key to open the doors. Thus, the car is prevented from reading the signal but the attackers can receive it by using an appropriate frequency. Once the attackers acquire the code, they could stop jamming and start replaying, but if the user does press the key a second time (which is very frequent), a new code will invalidate the previous code held by the attackers. That is why the attackers will not stop jamming at the first code but wait for the user to press the key again to have the second rolling code. They will then replay the first code, which has not been invalidated since the second code did not reach the car because of jamming, so the user can get into the car. The second code is set aside and will be used later for carrying the attack.

So let P be the set of atomic attacks associated with the scenario of the attack based on the rolling code. We have identified the following predicates:

- $P1$: The owner of the vehicle emits a rolling code signal to unlock the car by pressing a button;
- $P2$: The attacker jams the radio signal from the remote key;
- $P3$: The attacker achieves a radio signal interception and records the first rolling code;
- $P4$: The driver presses the key again;
- $P5$: The attacker continues to jam the signal emanating from the remote key;
- $P6$: The attacker executes again a radio signal interception to record the second rolling code;
- $P7$: During the user's second attempt, the first saved code is transmitted by the attacker to open the car;
- $P8$: The central locking system verifies the validation of the code to ensure that the code is correct and had not expired;
- $P9$: The car doors are unlocked;
- $P10$: The recorded second signal is set aside by the attacker for a later replay attempt.

Temporal operators allows the chronological order of attacks to be represented. So, we can gather for example the first three predicates ($P1$, $P2$ and $P3$), followed by the next three predicates ($P4$, $P5$ and $P6$) in the next instance (AX). Once the second code is intercepted, the attacker can replay the first code in the next step and the car will be unlocked instantly ($P7$, $P8$ and $P9$) . $EF\ P10$ means that there exists a path that $P10$ will be true in the future. Therefore, another replay attempt can be realised to take over the car. Figure 6 represents the modeling of the attack with the extended attack tree.

4.2 | System Modeling

The attack environment with different actors, namely the user and the attacker, and system components is given in Figure 7. The key fob is controlled physically by the driver, through pressing on the unlock button. The remote key transmits a radio signal to a micro-controller inside the car that will accept or deny to open the door, according to the rolling code validation. The central locking system containing the micro-controller is connected to the doors via a CAN BUS protocol. The attacker has a physical connection with the jammer to scramble the signal emitted from the remote key. He also uses the frequency repeater (replayer) to send radio signals to the central locking system of the car. The different rolling codes are saved in the recorder in order to be replayed later. Our Connectivity Profile is used to model the technology properties such as the range of the radio transmitter (5 to 20 meters), the allowed frequency, etc.

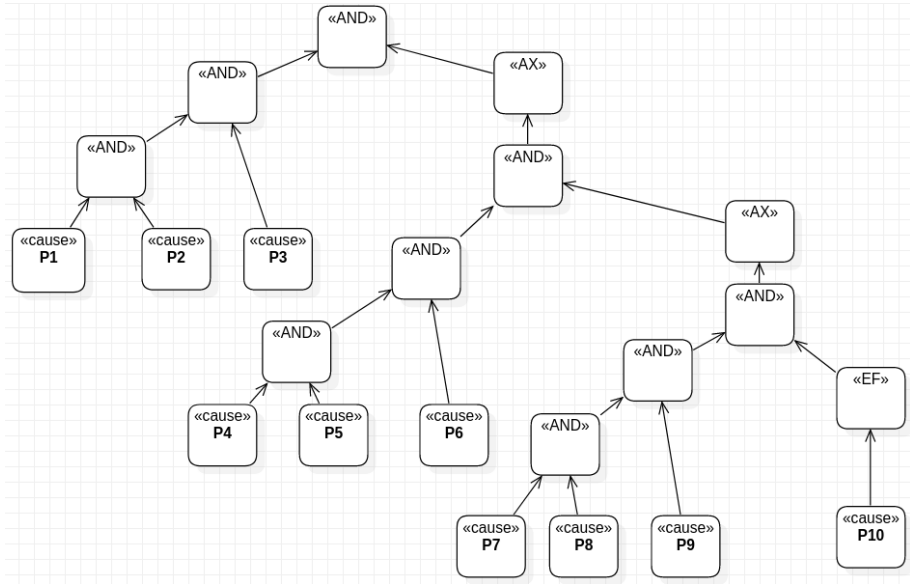

FIGURE 6 Extended Attack Tree for Rolling Code Attack

For the dynamic behavior of the system, we use two state machine diagrams : one for the whole system and one for the central locking sub-system. The remote key generates a coded signal which, when recognized by the control unit, allows the doors to be locked or unlocked. This control unit locks or unlocks the door only when the correct bit sequence is detected. We assume in this example the use of TEA5500 type coding, which consists of a series of 24 bits. Therefore, the decoder receives 6 digits in hexadecimal, for example the combination "F4ABDC". Figure 8 represents the seven states of the automaton. At the beginning, the control unit waits in state $S1$ for processing the first series of 4 bits transformed by the signal and which corresponds to an F in hexadecimal. This is why we put the transition $T1$ labeled "0-1-2-3-4-5-6-7-8-9-A-B-C-D-E" which loops the initial state on itself: as long as the user does not type an F, the system state does not change. Then, in state $S2$, we know that the digit F is transmitted, but as long as it is not followed by a 4, the system does not go to state $S3$. The transition system continues to react according to the received signal (code). Ultimately, either it evolves towards the $S7$ state to activate the opening, or it returns to the $S1$ state for a new capture.

Figure 9 represents the different states of the whole system, i.e. our system of interest, and the transitions between states. The system covers 6 states : $SS1$, $SS2$, $SS3$, $SS4$, $SS5$ and $SS6$. $SS1$ represents the initial state, in which there is no attack and the doors are closed. In a normal mode, if the user presses the key to unlock the doors, the system will pass to $SS2$, and if he presses the closing button, it will come back to $SS1$. However, an attacker can jam the emitted signal and intercept the opening code, and in this case, the system falls in $SS3$. When the user presses the key again, the second code will be recorded as in state $SS4$. If the jamming is stopped, depending on the current state $SS3$ or $SS4$, there will be a transition to $SS2$ to open the doors by using the non-expired code. If not, the attacker can replay the first code, the doors will be unlocked but the system is compromised ($SS5$). Later, a replay of the second code will induce an attack on the system, a state that is identified by $SS6$. Since we are interested in the taking over of the car, the system states in this diagram will be used to verify if the system is protected against the attack.

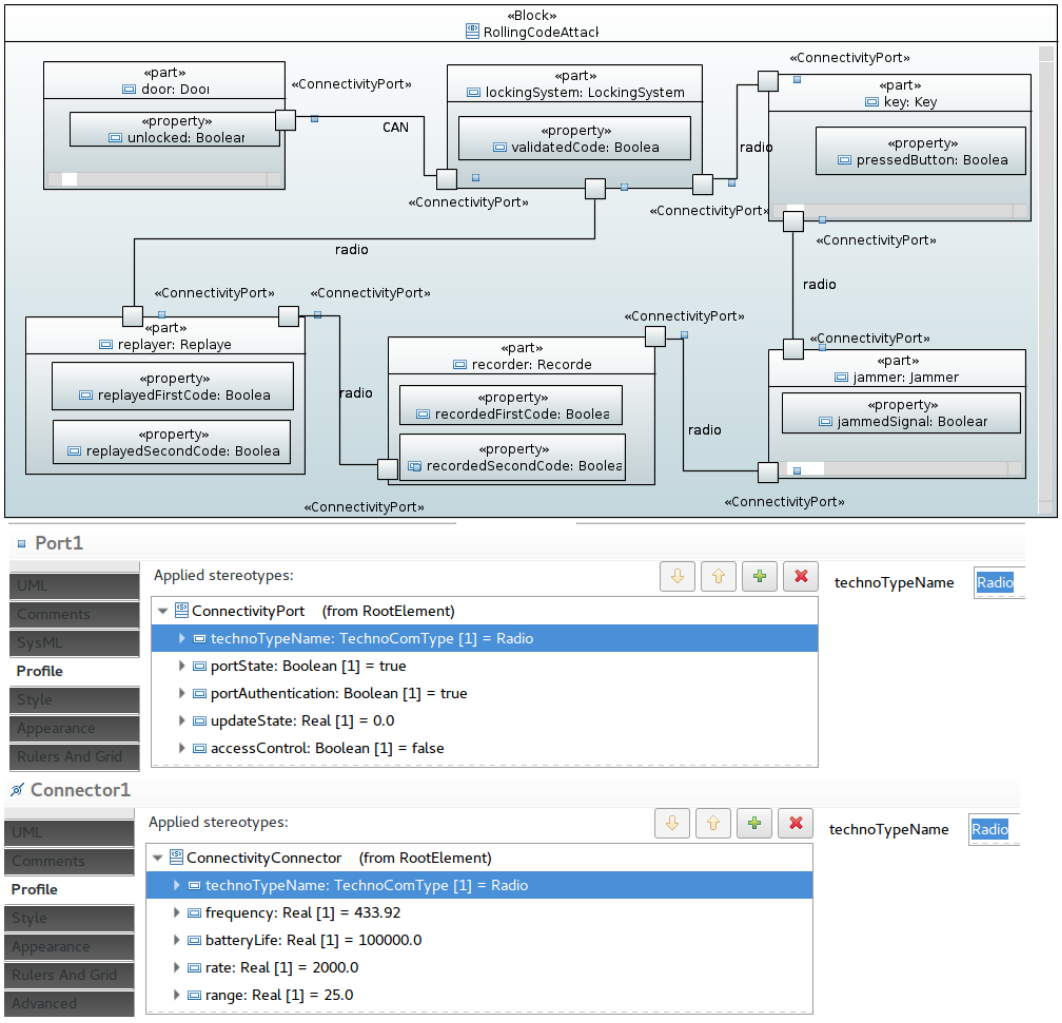

FIGURE 7 System Internal Block Diagram

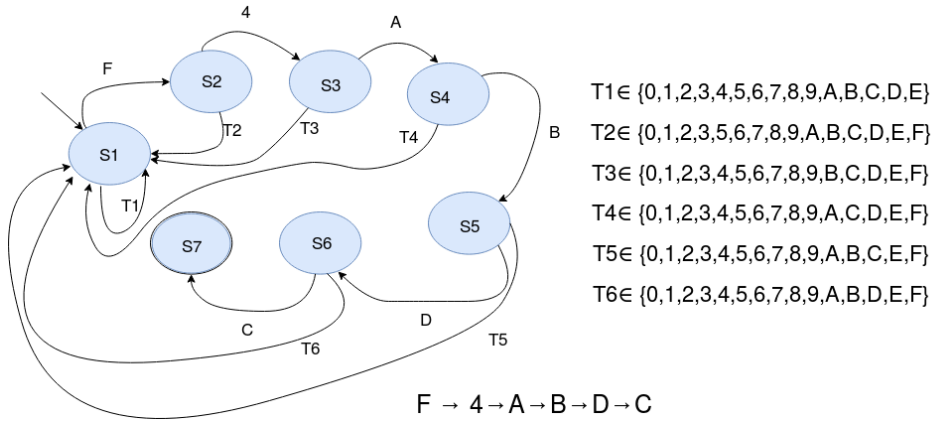

FIGURE 8 Locking System State Machine Diagram

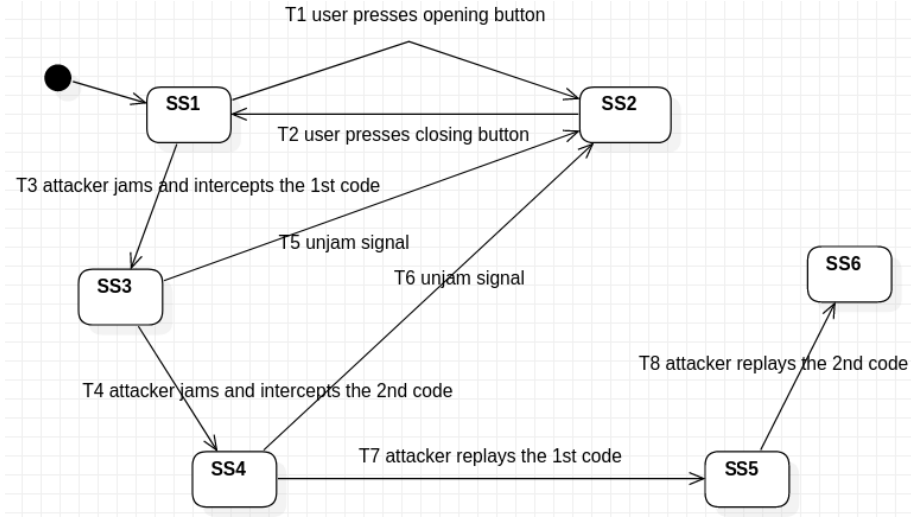

FIGURE 9 System State Machine Diagram

4.3 | Model Transformation and System Validation

Once the CIM and PIM models have been refined, we can apply model transformations described in Section 3.3 to generate model checking code. In order to generate an executable code, an additional step has to be carried out to refine the attack predicates. We need to map the atomic attacks into system attributes that correspond to variables in the NuSMV code. These attributes have also been modeled in the previous IBD (Figure 7). Table 3 represents the mapping between the predicates and the system values.

TABLE 3 Attack Predicates and System Attributes Mapping

Predicate	Description	Attribute
P1	Press the unlock button	key.pressedButton
P2	Undergo a radio signal jamming attack	jammer.jammedSignal
P3	Undergo a radio signal interception for the 1st code	recorder.recordedFirstCode
P4	Press the unlock button again	key.pressedButton
P5	Undergo a radio signal jamming attack	jammer.jammedSignal
P6	Undergo a radio signal interception for the 2nd code	recorder.recordedSecondCode
P7	Replay the 1st code	replayer.replayedFirstCode
P8	Verify the code validation	lockingSystem.validatedCode
P9	Unlock car doors	door.unlocked
P10	Replay the 2nd code later	replayer.replayedSecondCode

The code below gives an extract from the generated NuSMV code, with the CTL specification corresponding to the extended attack tree in Figure 6.

```
MODULE Key
VAR
   pressedButton : boolean;
ASSIGN
   init(pressedButton) := FALSE;
   next(pressedButton) := case
      pressedButton = TRUE : FALSE;
      TRUE : {TRUE,FALSE};
   esac;
MODULE Jammer [...]
MODULE Recorder [...]
MODULE Replayer [...]
MODULE LockingSystem [...]
MODULE Door [...]
MODULE main
VAR
   key: Key;
```

```
   jammer: Jammer;
   recorder : Recorder;
   replayer : Replayer;
   lockingSystem : LockingSystem;
   door : Door;
   state : {SS1, SS2, SS3, SS4, SS5, SS6};
ASSIGN
   init(state) := SS1;
   next(state) := case
      -- T3 From SS1 to SS3 -----------------------------
      state = SS1 & key.pressedButton & jammer.jammedSignal & recorder.recordedFirstCode : SS3;
      -- other transitions ...
      TRUE : state;
   esac;
DEFINE
   P1 := key.pressedButton;
   -- P2 to P10 ...
SPEC AG !(P1 & P2 & P3 & (AX (P4 & P5 & P6 & (AX (P7 & P8 & P9 & (EF P10))))))
```

By running the generated NuSMV code and the specification *SPEC AG !(RollingCodeAttack)*, there exists a path through which the attack can take place as shown in the counterexample in Figure 10. At the beginning, the system is in a nominal state, with the doors closed. Then, the user presses the key while the signal is being jammed by the attacker, allowing the latter to intercept the first code. When the user presses again the key, the attacker records the second code. The attacker will then replay the first code, which is still valid to open the doors, henceforth compromising the system. Later on, he can replay the second code to open the car, showing the weakness of the system. As recommended in [34], different solutions could be used to protect systems against this kind of attack, such as adding a challenge/response via transceivers instead of the one-way communication between the remote controller and the micro-controller, or using time-based algorithms to protect against unwanted replay.

5 | CONCLUSION

In this paper, we have proposed MBSAES, a model-based approach for security analysis consisting of three steps : i) conceptual modeling of the security requirements, ii) conceptual modeling of the system, and iii) transformation of the models for verification. These three steps are mapped to MDA concepts, by separating the conceptual modeling from the technical platform. The former allows the manipulation of an intermediate model which facilitates communication between the project manager and the client, while the latter carries out the formal verification. We have thus developed semi-formal models which do not carry many ambiguities for the expression of needs and which are relatively easy to interpret. The SysML language was chosen because of its adaptability and its extensibility by profiling. A profile called Extended Attack Tree has been introduced, allowing a formal and temporal modeling of attacks. A Connectivity Profile has also been proposed to enable the propagation of flows between the different communicating subsystems which are most often heterogeneous. Within the defined framework, we have developed two transformation processes following the MDA paradigm. The first one ensures a mapping from an Extended Attack Tree to a CTL formula, by using a Java program. The second one aims at generating NuSMV code from SysML models using a transformation

```
-- specification AG !(((P1 & P2) & P3) & AX (((P4 & P5) & P6) & AX (((P7 & P8) & P9) & EF P10)))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    key.pressedButton = FALSE
    jammer.jammedSignal = FALSE
    recorder.recordedFirstCode = FALSE
    recorder.recordedSecondCode = FALSE
    replayer.replayedFirstCode = FALSE
    replayer.replayedSecondCode = FALSE
    lockingSystem.validatedCode = FALSE
    door.unlocked = FALSE
    state = SS1
  -> State: 1.2 <-
    key.pressedButton = TRUE
    jammer.jammedSignal = TRUE
    recorder.recordedFirstCode = TRUE
    state = SS3
  -> State: 1.3 <-
    key.pressedButton = FALSE
  -> State: 1.4 <-
    key.pressedButton = TRUE
    recorder.recordedSecondCode = TRUE
    state = SS4
  -> State: 1.5 <-
    key.pressedButton = FALSE
    replayer.replayedFirstCode = TRUE
    lockingSystem.validatedCode = TRUE
    door.unlocked = TRUE
    state = SS5
  -> State: 1.6 <-
    replayer.replayedFirstCode = FALSE
    replayer.replayedSecondCode = TRUE
    state = SS6
```

FIGURE 10 Rolling Code Attack Counter Example

by template with Acceleo. The hence generated models are checked on the NuSMV model checker. Thanks to this, we can accurately detect the conditions that led to an attack if the system is not protected against it. With respect to the addressed research question, the processes in our hybrid methodology, combining semi-formal modelling and formal verification, can be done automatically and early, which will facilitate the secure-by-design approach. We have applied MBSAES to two well-known cyber-attacks, perpetrated on a Jeep Cherokee in 2014 and on a Tesla Model S in 2016 [6], respectively. These different complex examples provide strong support for our method, and actual design errors were detected with counterexamples. This aligns with the intended outcomes of the Systems Security Engineering working group [35] from the International Council on Systems Engineering (INCOSE) whose mission is to provide means and methods for enabling and facilitating effective system security analysis in systems engineering.

The present work is meant to be extended further in the following directions. First, we need to work on a benchmark that allows evaluating MBSAES systematically, on the way it helps to reduce late errors, development time and system complexity via the security by design paradigm and the abstraction and automation mechanisms. In addition, we contemplate the construction of a library of attack patterns as well as attack compositions to represent the combination and propagation of errors. This database will be generated from existing vulnerabilities on connected objects coming from for example the MITRE Common Weakness Enumeration (CWE) and the Common Attack Pattern Enumeration and Classification (CAPEC) dictionaries [36]. Attack patterns will be modeled with CTL by using our Extended Attack Tree model. Although temporal logic specifications are well suited to model attack scenarios with a stepwise execution of actions, it is noted that the formalism should be enriched to model the parallel behaviour of attacks. This database of extended attack trees with temporal operators can be built automatically and systems will be validated against these attacks. In this way, we can detect rapidly the security flaws during the design phase, based

on existing vulnerabilities on connected systems. In a system-of-systems perspective, this allows to choose the best solutions among diverse system architecture candidates, from the security viewpoint. Besides, our methodology will be further extended with the arrival of SysML v2 [37] for which the verification and validation of complex systems will be one of the future pillars. A more formal specification of SysML syntax, semantics and the mappings between them, by using a declarative approach, will reduce the gap between this systems modeling language and a formal verification language.

References

[1] MDA Guide Version 1.0.1. Object Management Group; 2003.

[2] MDA Guide rev. 2.0. Object Management Group; 2014.

[3] Czarnecki K, Helsen S. Classification of model transformation approaches. In: 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, vol. 45 Anaheim, USA; 2003. p. 1–17.

[4] Mens T, Van Gorp P. A taxonomy of model transformation. Electronic Notes in Theoretical Computer Science, International Workshop on Graph and Model Transformation 2006 March;152:125–142.

[5] Blanc X. MDA en action Ingénierie Logicielle Guidée par les Modèles. Eyrolles; 2005.

[6] Mili S, Nguyen N, Chelouah R. Transformation-based Approach to Security Verification for Cyber-Physical Systems. IEEE Systems Journal 2019 December;13:3989 – 4000.

[7] Systems Modeling Language version 1.6. Object Management Group; 2019.

[8] Cimatti A, Clarke E, Giunchiglia F, Roveri M. NuSMV: a new symbolic model checker. International Journal on Software Tools for Technology Transfer 2000;2(4):410–425.

[9] Nguyen P, Wang S, Yue T. Model-Based Security Engineering for Cyber-Physical Systems: A Systematic Mapping Study. Information and Software Technology 2016 11;83.

[10] Basin D, Doser J, Lodderstedt T. Model driven security for process-oriented systems. In: 8th ACM Symposium on Access Control Models and Technologies Como, Italy; 2003. p. 100–109.

[11] Jürjens J, Shabalin P. Automated verification of UMLsec models for security requirements. In: International Conference on the Unified Modeling Language Lisbon, Portugal: Springer; 2004. p. 365–379.

[12] Holzmann GJ. The Model Checker SPIN. IEEE Transactions on Software Engineering 1997 May;23(5).

[13] Fernández-Medina E, Piattini M. Designing secure databases. Information and Software Technology 2005 May;47(7):463–477.

[14] Reznik J, Ritter T, Schreiner R, Lang U. Model Driven Development of Security Aspects. Electronic Notes in Theoretical Computer Science 2007;163:65–79.

[15] Sánchez O, Molina F, Garcıa-Molina J, Toval A. ModelSec: A Generative Architecture for Model-Driven Security. Journal of Universal Computer Science 2009;15(15):2957–2980.

[16] Delange J, Pautet L, Feiler P. Validating safety and security requirements for partitioned architectures. In: International Conference on Reliable Software Technologies Brest, France: Springer; 2009. p. 30–43.

[17] Feiler PH, Lewis BA, Vestal S, Colbert E. An Overview of the SAE Architecture Analysis & Design Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering. In: International Federation for Information Processing / Workshop on Architecture Description Languages Toulouse, France; 2004. .

[18] Ouchani S, Jarraya Y, Mohamed OA. Model-based systems security quantification. In: IEEE International Conference on Privacy, Security and Trust Montreal, Quebec, Canada; 2011. p. 142–149.

[19] Saadatmand M, Cicchetti A, Sjödin M. On the need for extending MARTE with security concepts. In: International Workshop on Model Based Engineering for Embedded Systems Design Newport Beach, USA; 2011. .

[20] Faugère M, Bourbeau T, Simone R, Gérard S. MARTE: Also an UML profile for modeling AADL applications. In: 12th International Conference on Engineering of Complex Computer Systems Auckland, New Zealand; 2007. p. 359–364.

[21] Apvrille L, Roudier Y. SysML-Sec attack graphs: compact representations for complex attacks. In: International Workshop on Graphical Models for Security Verona, Italy: Springer; 2015. p. 35–49.

[22] Larsen KG, Pettersson P, Yi W. UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer 1997;1(1-2):134–152.

[23] Roudier Y, Apvrille L. SysML-Sec: A model driven approach for designing safe and secure systems. In: IEEE International Conference on Model-Driven Engineering and Software Development Angers, France; 2015. p. 655–664.

[24] Blanchet B. Proverif automatic cryptographic protocol verifier user manual. CNRS, Departement d'Informatique, Ecole Normale Superieure, Paris 2005;.

[25] Li L. Safe and secure model-driven design for embedded systems. PhD thesis, Université Paris-Saclay; 2018.

[26] Mota A, Farias A, Woodcock J, Larsen P. Model checking CML: tool development and industrial applications. Formal Aspects of Computing 2015 09;27:975–1001.

[27] Wolper P. The tableau method for temporal logic: An overview. Logique et Analyse 1985;p. 119–136.

[28] Rose LM, Matragkas N, Kolovos DS, Paige RF. A feature model for model-to-text transformation languages. In: Proceedings of the 4th International Workshop on Modeling in Software Engineering IEEE Press; 2012. p. 57–63.

[29] Pudar S, A pragmatic method for integrated modeling of security attacks and countermeasures, Iowa State University, USA; 2007.

[30] MOF Model to Text Transformation Language. Object Management Group; 2008.

[31] Acceleo User Guide, https://wiki.eclipse.org/Acceleo/User_Guide;.

[32] Clarke EM, Grumberg O, Peled D. Model checking. MIT press; 1999.

[33] NuSMV: a new symbolic model checker, http://nusmv.fbk.eu;.

[34] Kamkar S. Drive it like you hacked it: New attacks and tools to wirelessly steal cars. DEFCON 2015;23.

[35] INCOSE System Security Engineering Working Group hiomrgse;.

[36] Common Vulnerabilities and Exposures. http://cve.mitre.org. The MITRE Corporation;, `http://cve.mitre.org/`.

[37] Systems Modeling Language (SysML) v2 Request For Proposal (RFP). Object Management Group; 2017.